VERSITA

Int. J. Appl. Math. Comput. Sci., 2012, Vol. 22, No. 2, 423–435
DOI: 10.2478/v10006-012-0032-7

amcs

A GENERAL ON–THE–FLY ALGORITHM FOR MODIFYING THE KINEMATIC TREE HIERARCHY

JAKUB STĘPIEŃ [*,**], ANDRZEJ POLAŃSKI [*,**], KONRAD WOJCIECHOWSKI [*,**]

[*] Institute of Informatics
Silesian University of Technology, Akademicka 16, 44-100 Gliwice, Poland
e-mail: {Jakub.Stepien,Andrzej.Polanski,Konrad.Wojciechowski}@polsl.pl

[**] Polish-Japanese Institute of Information Technology
Aleja Legionów 2, 41-902 Bytom, Poland

When conducting a dynamic simulation of a multibody mechanical system, the model definition may need to be altered during the simulation course due to, e.g., changes in the way the system interacts with external objects. In this paper, we propose a general procedure for modifying simulation models of articulated figures, particularly useful when dealing with systems in time-varying contact with the environment. The proposed algorithm adjusts model connectivity, geometry and current state, producing its equivalent ready to be used by the simulation procedure. Furthermore, we also provide a simple usage scenario—a passive planar biped walker.

Keywords: dynamics, articulated system, rigid bodies, system hierarchy, contact, animation.

1. Introduction

Modeling multibody mechanical systems subject to multiple and time-varying contacts with the environment is still a challenging problem for researchers designing procedures for simulating multibody mechanical systems. Attaching the system to a base can be used as a computationally cheap and precise way to simulate systems having a single contact point with the environment as long as we can express the nature of the contact with a bilateral constraint (*a joint*). Such a bilateral contact constraint is a reasonable approximation for a sticking contact, e.g., a human/robot leg placed on a *non-slippery* ground. However, expressing the single contact with the environment by means of a joint attached to a base makes this contact enter the model definition, and whenever the location of the contact changes (or vanishes) during the course of simulation, care must be taken to alter the model definition.

In this paper we describe linear-time, on-the-fly procedure for modifying the multibody mechanical system model definition applicable to kinematic trees, which modifies system topology, geometry and state, expecting minimal information from the user.

2. Background

In the classical approach to the problem of modeling and solving the dynamics of multibody mechanical systems, one needs to explicitly specify equations of motion in the form of systems of differential equations. These equations are solved by appropriate algorithms for integration. In the case of a system with bodies connected by multiple joints of different types, multiple configurations of external constraints and interactions with the environment, modeling a dynamical system can become difficult in the sense that determining both model equations and their parameters may require considerable research and computational effort.

In the course of the development of computer methods for multibody dynamic problems, alternative approaches have been proposed, which utilize various *dynamics algorithms* to either compute first derivatives of the state vector directly or to determine coefficients of equations of motions efficiently. Parameters of a dynamics algorithm are defined by the structure of the mechanical model considered and properties of the rigid bodies forming it. Thanks to such an approach, researchers can develop systems for automatic analysis of multibody mechanical systems, applicable to certain classes of systems

regardless of their exact structure. This methodology for modeling the dynamics of multibody mechanical systems will be called here the *algorithmic* or *algorithm-based* approach.

One of the earliest dynamic algorithms emerged in the mid-1960s with solutions based on both Newtonian (Hooker and Margulies, 1965; Roberson and Wittenburg, 1966) and Lagrangian (Uicker, 1965) mechanics. By the mid-1970s, solutions utilizing Lagrangian multipliers for joint constraints (Chace and Sheth, 1973; Orlandea *et al*., 1977) appeared. What is more, Orlandea's program (ADAMS) exploited matrix sparsity (sparse tableaux formulation) to improve performance. These early works are well reviewed by Paul (1975). The *Composite-Rigid-Body Algorithm* (CRBA), first proposed by Walker and Orin (1982), harnessed the $O(n)$ recursive Newton–Euler inverse dynamic algorithm (RNEA) as a means of calculating the joint-space inertia matrix of a kinematic tree much faster than earlier methods. The CRBA is most commonly presented as an $O(n^2)$ solution, but obviously once the entries of the inertia matrix have been determined, a system of linear equations needs to be solved, thus the dynamics algorithms using the CRBA are collectively referred to as $O(n^3)$ algorithms.

The earliest known forward dynamics algorithm having linear time complexity was proposed by Vereshchagin (1974), but it was not until Featherstone's works on the *Articulated-Body Algorithm* (ABA) (Featherstone, 1983; 1984; 1987) that propagation methods became widely acknowledged. An alternative linear time solution is due to Baraff (1996), who coupled an *always sparse* formulation using Lagrangian multipliers with linear-time factorization. This approach has been particularly popular within the computer graphics community, where rigid body dynamics algorithms are used to build general purpose physics engines and thus solutions based on Lagrangian multipliers, which provide natural means for mixing articulation and contact constraints, are preferred. A recent, thorough comparison of multibody dynamic algorithms can be found in the work of Jain (2011).

3. Contribution

The algorithmic approach to the modeling of multibody mechanical systems uses the topology of the system in question as input data. Based upon the information provided in the particular system representation, the algorithm traverses the hierarchy and applies equations governing the motion of its elements to eventually determine system accelerations either one-by-one (e.g., ABA) or simultaneously (e.g., CRBA).

System topology is usually expressed using a graph containing bodies interconnected by joints. The entire system is (topologically) attached to a referential body called a *base* or a *carrier*. The base is either immobile or its motion is known as a function of time. If there are no kinematic loops in the system, the graph becomes a tree rooted at the base.

In the case of a kinematic tree, the attachment of the system to a base can be used as a computationally cheap and precise way to simulate systems having a single contact point with the environment as long as we can express the nature of the contact with a bilateral constraint (a *joint*). Apart from systems which are permanently attached to the ground/environment, the bilateral contact constraint is a reasonable approximation for a sticking contact, e.g., a human/robot leg placed on a *non-slippery* ground. However, expressing the single contact with the environment by means of a joint attached to the base makes this contact enter the model definition, and, whenever the location of the contact changes (or vanishes) during the course of simulation, care must be taken to alter the model definition. Presumably, changing the model must be followed by an appropriate adjustment of its state to achieve equivalent configuration.

This kinematic tree hierarchy modification paradigm can entirely eliminate the need of equipping a generalized coordinate algorithm with the ability to handle contact constraints if during the analyzed motion the system is guaranteed to be in one contact with the external environment at most (e.g., simplified bipedal walking, human running). In other cases, it can still be useful to decrease the total amount of contact constraints and thus the amount of necessary computations.

In this paper we describe a low-overhead, scalable, on-the-fly procedure for modifying kinematic tree hierarchy applicable to the algorithm-based approach for simulations of multibody mechanical systems. The model definition which we are concerned with is tightly based upon Featherstone's works on rigid body dynamics (Featherstone, 2008; 1987) and includes a connectivity graph and system geometry described in introductory sections. The rigid bodies themselves are defined by their masses and the moment of inertia tensors. All of these elements need to be redefined in order to properly modify the system hierarchy. The *on-the-fly* property of our procedure means that it can be applied without interrupting the simulation. Its scalability is due to favorable time complexity, which is *linear* in the number of bodies in the worst case.

Although the procedure was designed to be compatible with the model definition based on Featherstone's works which enforces a number of conventions, the general idea is applicable to any dynamics algorithm which is explicitly using system topology to traverse the tree. Such realizations will most naturally use recurrence relations and have state and geometry variables referred to link coordinates. Applying the proposed procedure to algorithms based on redundant set coordinates which constrain the motion of the bodies with reaction forces does not seem sensible since the role of explicit topological information

is such cases is only supplementary or none.

When designing the hierarchy switching algorithm we have made several assumptions as to what qualities it should possess:

- **A1:** The operation should be revertible (*i.e.* , $op(op(model))= model$).
- **A2:** The operation should result in a geometrically equivalent representation (system *looks* the same).
- **A3:** The operation should result in a representation physically equivalent *at the instant of applying it*.

These assumptions will be referred to throughout the text.

The following three sections provide a brief overview of the algebraic prerequisites and model elements but are not intended as a comprehensive discourse. For a more thorough and complete description, please refer to Featherstone (2008).

4. Spatial algebra

Spatial algebra for multibody mechanical systems is exhaustively discussed in prior publications (Featherstone, 2008; 1987; Rodriguez, 1991). We use the algebraic approach shown there. Some notational conventions and basic prerequisites, which will enable the reader to understand our ideas, are presented below.

Thanks to spatial algebra, the traditional separate treatment of the linear and angular aspects of dynamics are merged into a uniform notational form consisting of both. Instead of a pair of 3D Euclidean vectors we can use a single 6D spatial motion vector (from M^6 space) or a 6D spatial force vector (from F^6 space):

- linear and angular velocities $\rightarrow$ spatial velocity,
- linear and angular accelerations $\rightarrow$ spatial acceleration,
- momentum and angular momentum $\rightarrow$ spatial momentum,
- force and moment of force (couple) $\rightarrow$ spatial force.

Furthermore, the *spatial inertia tensor* of a rigid body relates its *spatial* velocity and momentum, therefore it needs to bear information about the body's rotational inertia along with its mass. It is represented by a 6×6 matrix (called a *matrix of inertia* throughout the text).

Coordinate transforms are also expressed using 6×6 matrices (which we will generally denote by $\boldsymbol{X}$) comprising both rotation and translation. The vector spaces M^6 and F^6 are dual, so there is a scalar product defined between them which implies the following relation:

$$\boldsymbol{X}^* = \boldsymbol{X}^{-T}, \tag{1}$$

where both $\boldsymbol{X}$ and $\boldsymbol{X}^*$ perform the same coordinate transformation but on vectors from M^6 and F^6, respectively.

For convenience and brevity we shall assume that we have several auxiliary procedures at hand:

- $rot(\boldsymbol{E})$: returns a spatial transform built from the input 3×3 rotation matrix $\boldsymbol{E}$;
- $trn(\boldsymbol{t})$: returns a spatial transform built from the input 3×1 translation vector $\boldsymbol{t}$;
- $ori(\boldsymbol{X})$: returns a 3×3 rotation matrix implicit in the spatial transform $\boldsymbol{X}$;
- $pos(\boldsymbol{X})$: returns a 3×1 translation vector implicit in the spatial transform $\boldsymbol{X}$.

If we ever need a position of the origin of the coordinate frame represented by spatial transform $\boldsymbol{X}$, we will call $pos(\boldsymbol{X})$, and if we happen to need this frame's orientation, we call $ori(\boldsymbol{X})$. Here rot and trn perform *inverse* operations. Thus,

$$\boldsymbol{X} = rot(ori(\boldsymbol{X}))\, trn(pos(\boldsymbol{X})). \tag{2}$$

These functions can either launch apt computations each time they are called or refer to a precalculated matrices and vectors set/cache.

5. System model

The system model is provided by descriptions of its connectivity and geometry, the former providing us with information about the hierarchy and nature of inter-body connections within the multibody system while the latter—about locations of these connections.

5.1. Connectivity. We shall be using a representation proposed by Featherstone (2008), but it should be noted that equivalent alternatives can be found in the literature (Wittenburg, 2007). The connectivity (or topology) of the system is defined by a connectivity graph, which is

- undirected (i.e., the relation defined by the graph edge is symmetric) and
- connected (i.e., there exists a path between any two vertices).

Vertices in the graph represent bodies comprising the system, while edges—joints (inter-body connections). The notion of a graph is used to allow expressing kinematic loops in the system by cycles in the graph. However, the analysis provided in this paper does not consider kinematic loops so we can restrict the graph to be a topological tree (called simply a *tree*). Moreover, for a system with kinematic loops it is reasonable to set off with its spanning tree and then apply additional loop-closure

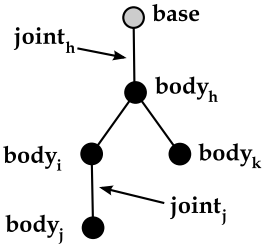

Fig. 1. Sample connectivity graph. Two of the joints are labeled to show where they are in the graph and how they are numbered. There are no cycles in the graph so it is a topological tree.

constraints so restraining the analysis to a tree is justified (Wittenburg, 2007; Featherstone, 2008). The tree is rooted at the fixed base, which will normally have only one child (we will refer to it as the *top* or the *oldest* vertex/node), since the base can be regarded as having infinite mass, which means that subtrees rooted at its direct successors will not have influence on each other (unless a collision between them occurs).

Each vertex in the tree has a unique index number assigned to it. The index of the oldest/top vertex in the hierarchy is denoted by i_{top}. Edges are numbered so that edge i connects vertex i with its direct predecessor (*parent*) (Fig. 1).

Featherstone suggested the use of a parent array, λ, to store the connectivity information, which also implies a specific numbering of the bodies (Featherstone, 2008). The parent array is constructed in such a way that its i-th entry holds the index of the parent of body indexed with i. However, we have found an *object-oriented* approach by far more scalable and easy to manage, especially when switching between different hierarchies. By 'object-oriented' we mean holding separately for each joint the indices of

- its parent,
- its first child,
- its next sibling,

and in the algorithms using recursion rather than loops. Nevertheless, we shall use the symbol $\lambda(i)$ to denote an index of the parent of body i, but bear in mind that it is not an actual array any more (Fig. 2).

5.2. Joint models. A joint is a linking point between a parent and child body in the tree and it defines their relative motion freedom. In each such relationship, we shall call the parent body a *joint owner*, and the child body a *joint user*. Each body can *own* zero or more joints and must *use* exactly one. Note that, due to the edge and vertex numbering scheme described in the previous

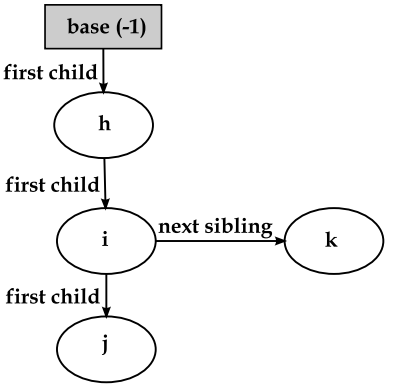

Fig. 2. Sample hierarchy. Each node's children and siblings are pointed to by accordingly labeled arrows, while the parenting goes as follows: $\lambda(h) = -1, \lambda(i) = h, \lambda(k) = h, \lambda(j) = i$.

section, each joint bears the same index as its user.

The joints are described by

- motion subspace S (given by a matrix $\boldsymbol{S}$ such that range($\boldsymbol{S}$) $= S$) to which the relative motion of the successor is constrained, and
- state variables expressing joints' configuration and velocity in allowable motion subspace at the current instant (relative to the parent).

The more intuitive (and more often met in the literature (Craig, 2005)) single-DOF, Euclidean equivalent of the motion subspace matrix is the *joint axis*.

From the definition of the motion subspace matrix $\boldsymbol{S}$, one can write

$$\boldsymbol{v}_i = \boldsymbol{S}_i \, \dot{\boldsymbol{q}}_i, \tag{3}$$

where $\dot{\boldsymbol{q}}_i$ is the joint space velocity vector for body i, while $\boldsymbol{v}_i$ is its spatial velocity. The matrix $\boldsymbol{S}$ will differ depending on the joint type—for a revolute joint (also known as a hinge or pin joint), it is simply

$$\boldsymbol{S} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}^T, \tag{4}$$

which means that it allows only one degree of freedom associated with rotation about the body-local z-axis. Representing matrices $\boldsymbol{S}$ in body-local coordinates results in their simple form, which for a majority of joint types remains constant during the simulation (Featherstone, 2008).

For brevity, we adopt Featherstone's idea of the *jcalc* procedure defined for every joint type/model. The main purpose of calling *jcalc* is to convert joint state variables (specific to a joint model) to a general 6×6 spatial transformation matrix describing the transformation introduced by this joint (see the next section).

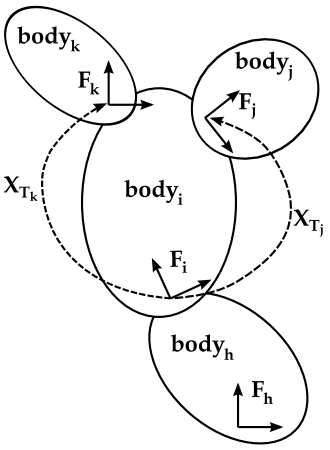

Fig. 3. Tree transforms for a sample system setting (2D for clarity). Origins of the bodies frames F_j and F_k are given relative to their parent (body i) by transforms $\boldsymbol{X}_{T_j}$ and $\boldsymbol{X}_{T_k}$, respectively.

5.3. Geometry. Having established the order and nature of connections between bodies comprising the system, we need to state *where* the joints are located and what geometric transformations they imply on bodies at the current instant. For the sake of clarity we now introduce the notion of a *pivot point*—the location of the joint that the i-th body stems out from. We shall use $\boldsymbol{p}_i$ to denote the pivot point for body *i*.

Body i has a coordinate frame F_i rigidly attached to it. We assume that F_i originates at $\boldsymbol{p}_i$. This is very important since it unifies each joint location with the origin of the frame of its *user*—a fact heavily exploited in Featherstone's dynamics algorithms. Although the efficiency and notational gains seem to promote this tight bound between a joint and its user, one cannot forget that each joint needs two bodies to ever exist.

The location of the pivot point relative to the body's Center Of Mass (COM) affects this body's moment of inertia tensor, which is expected to express the rotational inertia of the body when rotating about its pivot point. This COM to a pivot vector for the i-th body will be denoted by $\boldsymbol{r}_i$.

A body stems out from the joint used on the one hand, but can be a starting point for a number of subtrees rooted at child joints it is the owner of, on the other. The origin and orientation of the i-th child frame relative to the parent body's coordinate frame is given by the so-called *tree transform* denoted by $\boldsymbol{X}_{T_i}$ (Fig. 3):

$$\boldsymbol{X}_{T_i} = rot(\boldsymbol{E}_i)\, trn(\boldsymbol{t}_i), \tag{5}$$

where $\boldsymbol{E}_i$ and $\boldsymbol{t}_i$ are the i-th body orientation and translation relative to its parent, respectively, when the system is in the bind pose.

Tree transforms are enough to represent the tree configuration when all joint position variables are zero, which makes them an intuitive equivalent of the well-known computer animation notion—a bind pose, i.e., the starting/default configuration of the system. However, when joints' position variables differ from zero, we need a way to incorporate them into the geometric configuration. This is achieved by one more transformation, namely, *joint transformation*, which is denoted by $\boldsymbol{X}_{J_i}$ for the i-th joint. The method of converting joint-space position variables of a specific joint to a spatial transform $\boldsymbol{X}_{J_i}$ depends on the joint model, and we assume that it is provided by the aforementioned procedure *jcalc* available for each joint type.

We shall now show what *jcalc* does using a revolute joint. Since this kind of joint has only one degree of freedom (rotational about a fixed axis), it will hold only one internal state position variable, q_0, expressing the rotation across this joint. Let us assume that the rotation axis for this joint is Z. Thus, the coordinate transformation associated with this joint can be expressed by a 3×3 rotation matrix:

$$\boldsymbol{E} = \begin{bmatrix} \cos(q_0) & \sin(q_0) & 0 \\ -\sin(q_0) & \cos(q_0) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{6}$$

therefore the spatial joint transform for the i-th revolute joint is

$$\boldsymbol{X}_{J_i} = rot(\boldsymbol{E}), \tag{7}$$

and this is what the *jcalc* for this type of joint would return.

Now, using the tree transform (the bind pose, if you will) and the joints' state information, we can describe the geometry of the entire system in the *absolute* coordinate frame (base frame). Let $\boldsymbol{X}_i^{abs}$ be the coordinate transformation from the absolute frame to the i-th body's frame. Then

$$\boldsymbol{X}_i^{abs} = \boldsymbol{X}_{J_i}\, \boldsymbol{X}_{T_i}\, \boldsymbol{X}_{\lambda(i)}^{abs}, \tag{8}$$

which means that for body i we first apply the absolute transformation of the parent, $\boldsymbol{X}_{\lambda(i)}^{abs}$, then the transformation from the parent ($\lambda(i)$) frame to the child (*i*) frame and finally the state local transformation.

This concludes the model description part of this work. In the following sections, a procedure which coherently alters system connectivity and geometry is introduced along with a number of notions and definitions which emerged when designing it.

6. Algorithm

6.1. Modifying the connectivity. The connectivity information alone can be modified pretty easily, especially when the aforementioned object-oriented approach is used to represent the hierarchy. The procedure expects a single parameter—the index of the new top vertex in the tree, ${}_n i_{top}$. Once it is specified, we move along the path from

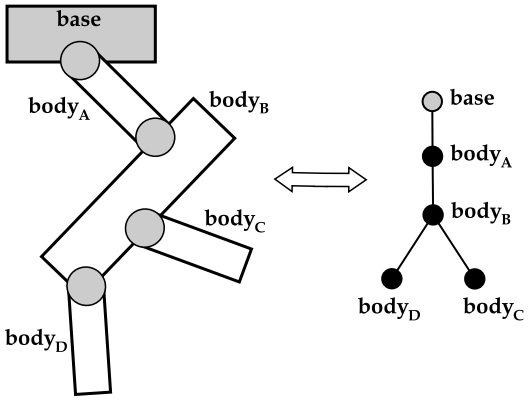

Fig. 4. Illustration of a simple multibody system. Left: system geometry (gray circles represent joints), right: corresponding connectivity tree.

the new oldest vertex to the previous one, inverting what one may call the polarity of connections.

For an unbranched system (*chain*) there is only one sensible way this can be done—every predecessor–successor relationship gets inverted. A choice needs to be made, however, on how to *reinterpret* ramification if the system in question is an actual tree. In such a case the chain of vertices connecting the new top body to the old one is isolated first. We shall call this chain a *chain of inversion*, χ. The parent–child relation along χ is subject to the same simple inversion rule as an unbranched system.

What about the siblings of a vertex laying along the chain of inversion, v_χ? We have considered two possibilities:

(i) the past siblings of v_χ remain its siblings in the new hierarchy (i.e., they change their parent to the new parent of v_χ);

(ii) the past siblings of v_χ keep their parent from the old hierarchy, thus becoming v_χ's *grand-children*.

If the first approach is applied to a system where the new top vertex has one or more siblings, we get several separate hierarchies attached at the base. This may not seem a great problem when considering the topological information itself, but one needs to remember that, as mentioned in Section 5.1, during simulation the base *isolates* its direct children from each other, which results in several sub-models functioning separately. The second approach is free from this problem since it generates topologies with a single child of the base (single top vertex), and this is why we chose it.

6.2. Modifying the geometry. Once we have defined what the new hierarchy is, we need to modify the geometric model so that the destination system has the same bind pose as the source one (A2). Since all the transformations kept are relative to the parent coordinate frame, they are no longer valid once the connectivity is changed. What persists is the absolute transformations, $\boldsymbol{X}_i^{abs}$.

However, in order not to violate A1, we need a way to isolate the geometric model from the state of the system so that on reverting the switching operation we get the original model and state. This cannot be achieved using absolute transformations only, which blend the state and the bind pose indistinguishably.

6.2.1. Bind pose. To reconstruct the bind pose alone, we need to calculate *absolute tree transforms*, $\boldsymbol{X}_{T_i}^{abs}$, as follows:

$$\boldsymbol{X}_{T_i}^{abs} = \boldsymbol{X}_{T_i}\, \boldsymbol{X}_{T_{\lambda(i)}}^{abs}, \tag{9}$$

which is Eqn. (8) after removing joint transforms. The orientations and origins/positions of body frames representing the system in its bind pose will be preceded by a *bind-* prefix (e.g., bind-orientation).

Now, we need to answer the question what exactly we need to find in order to reconstruct the bind pose. It is defined by tree transforms, $\boldsymbol{X}_{T_i}$, which in turn are defined by i-th joint's translation $\boldsymbol{t}_i$ and rotation $\boldsymbol{E}_i$ relative to its parent's frame (Eqn. 5). So for each joint we need to find the *new* $\boldsymbol{t}_i$ and $\boldsymbol{E}_i$, namely, ${}_n\boldsymbol{t}_i$ and ${}_n\boldsymbol{E}_i$.

Having found the building blocks of our new bind pose, we need to remember that bodies in the system have their rotational inertias expressed about pivot points, which have just been relocated. Thus, for each body we also need to find new pivot location relative to its COM, ${}_n\boldsymbol{r}_i$, and use it to recompute their inertial properties.

The procedures for tree transforms meant for the new top body and the remaining bodies are different, so they will be discussed separately in the following two sections.

Tree transforms: Non-top bodies. We have based this part of our algorithm on the observation that every inverted body–child relation in the connectivity model switches the owner–user roles around the linking joint. So, each joint remains at the same spot and preserves its properties, but it is *used* by a different body—its former owner. The former user, on the other hand now becomes the owner (Fig. 5).

Recall that we have assumed a convention in which a joint shares an index with its user. Since rearranging the connectivity tree changes joint users, it makes the indexing outdated, so we need an additional step to fix this. The effect that this correcting step has in terms of our notation is simply making the indices match again. What it actually means in terms of the underlying code depends entirely on implementation details.

Summarizing, what we need to do is, for the joints along χ,

(i) calculate absolute bind transforms for the old hierarchy, $\boldsymbol{X}_{T_i}^{abs}$;

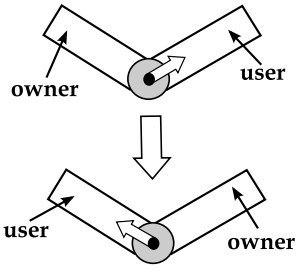

Fig. 5. Switching the user–owner relation around the joint. This is what happens along the chain of inversion as a result of changing the connectivity. As we can see, the joint is unaffected, it is just used/owned by a different body now.

(ii) change the connectivity;

(iii) update the joint-user correspondence to follow the connectivity change.

Now, we will express absolute tree transform frames of the bodies in the new hierarchy, ${}_n\boldsymbol{X}^{abs}_{T_i}$. The absolute bind-orientation remains obviously the same since it expresses how the body is oriented and it does not change. As for the frame absolute bind-origin, it is given by the position of a joint used by each body and this has just changed. As stated before, bodies will now *use* a joint they formerly *owned*. Although each body can own many joints, this still identifies the new joint uniquely since we are constrained to the chain of inversion. Summarizing, after the switch, bodies *use* joints that were *previously* used by their *current* parents (Fig. 6). Thus

$$ {}_n\boldsymbol{X}^{abs}_{T_i} = rot(ori(\boldsymbol{X}^{abs}_{T_i}))\, trn(pos(\boldsymbol{X}^{abs}_{T_{n\lambda(i)}})). \quad (10)$$

What we have is the absolute origin and orientation of i-th body frame, ${}_nF_i$, but what we really need is ${}_n\boldsymbol{t}_i$ and ${}_n\boldsymbol{E}_i$. Let us start with ${}_n\boldsymbol{E}_i$:

$$ {}_n\boldsymbol{E}_i = ori({}_n\boldsymbol{X}^{abs}_{T_i})\, ori({}_n\boldsymbol{X}^{abs}_{T_{n\lambda(i)}})^T, \quad (11)$$

which means that we calculate the new relative orientation of each body by *reducing* it by its new parent absolute orientation (this is why the transpose/inversion appears).

To position the frame origin relatively to its new parent frame, we apply

$$ {}_n\boldsymbol{t}_i = {}_n\boldsymbol{E}_{n\lambda(i)}\, (pos({}_n\boldsymbol{X}^{abs}_{T_i}) - pos({}_n\boldsymbol{X}^{abs}_{T_{n\lambda(i)}})). \quad (12)$$

First, the relative bind-position is found in absolute coordinates and then transformed to ${}_nF_{n\lambda(i)}$: the new frame of the i-th body's new parent.

This procedure needs to be applied to all bodies along the chain of inversion (apart from ${}_ni_{top}$) and their immediate successors that are not part of χ.

An additional note needs to be made here: Eqn. (12) will not yield correct results for the direct children of the new top body. The reason behind this is the fact that, in order for the system to be properly positioned with respect to the base, the translational part of ${}_n\boldsymbol{X}^{abs}_{T_{n\lambda(i)}}$ needs to include both the tree and joint transforms, since the new top joint is now fixed and we cannot arrange it to translate into to the desired position as a result of applying its joint transform later on. At the same time, we need to relate ${}_n\boldsymbol{t}_i$ to the bind-pose position of body $\lambda(i)$. This problem is solved by introducing an auxiliary vector variable, $\boldsymbol{g}_T$, presented in the next section.

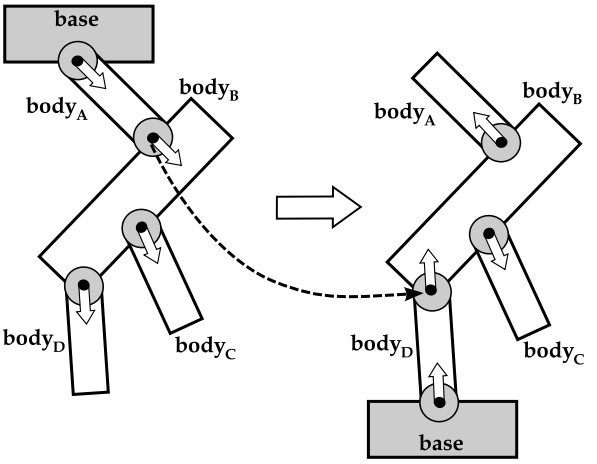

Fig. 6. Illustration of how the joint ownership changes along χ when the hierarchy is switched—bodies use joints that were previously used by their current parents. Solid, white arrows coming out of each joint indicate its polarity and point toward the body which stems out from it.

Tree transforms: New top-body. The new top body is the one that has come in contact with the base. This connection is entirely new in our system so we have no means of deducing anything about its nature. What we need from the user is to specify

- the nature of the new connection (i.e., joint type), and
- location of the new connection (let us denote it by $\boldsymbol{g}$).

The joint type does not influence our algorithm in any way since we assume that it is already designed to cope with different joint models. As for the location—it seemed reasonable to us that it would be specified by the absolute position. It is an intuitive algorithm parameter and should be easier to use with, e.g., third-party collision detection libraries/engines.

As previously, the absolute bind-orientation of this body remains the same. It is more difficult to get the new absolute bind-origin. We know the absolute location of the new top joint, $\boldsymbol{g}$, which includes both the bind pose and the current state of the system, but what we need is the absolute location of the new top joint when the system

is in the bind pose. Let us denote this point by $\boldsymbol{g}_T$. The procedure of determining $\boldsymbol{g}_T$ can be visualized as rigidly attaching a virtual marker to the new top body at $\boldsymbol{g}$ when the system is in its actual pose and then reconfiguring it into the bind pose. The final location of the virtual marker is the sought $\boldsymbol{g}_T$. It is achieved in the following way:

1. Find the translation vector (absolute coordinates) from the old hierarchy origin of this body's frame to the new origin, specified by the input parameter:
$$(*) = \boldsymbol{g} - pos(\boldsymbol{X}^{abs}_{{}_n i_{top}}) \tag{13}$$
2. Transform the translation vector $(*)$ from absolute coordinates to body coordinates—this vector is rigidly attached to the body frame so we will be able to rotate them together once expressed in body coordinates:
$$(**) = ori(\boldsymbol{X}^{abs}_{{}_n i_{top}})\,(*). \tag{14}$$
3. Transform the translation vector $(**)$ from body coordinates back to absolute coordinates, but using the tree transform this time:
$$(***) = ori(\boldsymbol{X}^{abs}_{T_{{}_n i_{top}}})^T\,(**) \tag{15}$$
4. Move the old hierarchy body frame absolute bind-origin by the absolute coordinates translation vector $(***)$:
$$\boldsymbol{g}_T = pos(\boldsymbol{X}^{abs}_{T_{{}_n i_{top}}}) + (***). \tag{16}$$

Equation (12) will use ${}_n\boldsymbol{X}^{abs}_{T_{{}_n i_{top}}}$ given by

$$rot(ori(\boldsymbol{X}^{abs}_{T_i}))\,trn(\boldsymbol{g}_T), \tag{17}$$

but the final model definition for the system should be based upon its proper form:

$$rot(ori(\boldsymbol{X}^{abs}_{T_i}))\,trn(\boldsymbol{g}). \tag{18}$$

The meaning of this is that in order to determine the tree transforms, we temporarily position the system at a virtual location, $\boldsymbol{g}_T$, which is uniquely defined by the bind-pose and the location of the old hierarchy top joint. Once the new tree transforms for all the bodies are found, we place the system at the correct location, $\boldsymbol{g}$, by providing a proper form of ${}_n\boldsymbol{X}^{abs}_{T_{{}_n i_{top}}}$.

The work in this section is complete, because the new top vertex has no parent, which means that we can use its freshly calculated absolute bind-orientation and bind-translation directly when defining the new tree transform for it.

Using ${}_n\boldsymbol{E}_i$, ${}_n\boldsymbol{t}_i$ and applying Eqn. (5), we can now calculate the new spatial joint transform, ${}_n\boldsymbol{X}_{T_i}$, for each joint.

Pivots with respect to body COMs. To determine the new moment of the inertia tensor of each body, we will need the new coordinates of the vector connecting its COM with the point about which this body pivots, ${}_n\boldsymbol{r}_i$. This vector is expressed in the i-th body's frame. Note, however, that we know nothing about the COM location apart from the old vector $\boldsymbol{r}_i$. Here is how it can be used:

$${}_n\boldsymbol{r}_i = \boldsymbol{r}_i + {}_n\boldsymbol{E}_i\,(pos({}_n\boldsymbol{X}^{abs}_{T_i}) - pos(\boldsymbol{X}^{abs}_{T_i})), \tag{19}$$

which means translating the old pivot location in accordance to how the joint was relocated after having expressed the relocation vector in the i-th body's coordinate frame.

Once again, a separate procedure is needed for the new hierarchy top joint. Since we have assumed a comfortable and intuitive parameter to be the new absolute top joint position, $\boldsymbol{g}$, we will have to work with the actual configuration of the tree rather than its bind-pose, but the formula is similar:

$${}_n\boldsymbol{r}_{{}_n i_{top}} = \boldsymbol{r}_{{}_n i_{top}} + ori(\boldsymbol{X}^{abs}_{{}_n i_{top}})\,(\boldsymbol{g} - pos(\boldsymbol{X}^{abs}_{{}_n i_{top}})). \tag{20}$$

6.2.2. Current configuration. The bind pose is already suited for the new connectivity, but the relative joint position variables represent the configuration of a different model, which in most cases will resultantly violate A2. This part can be done particularly conveniently and efficiently for non-top joints. The parent–child relations have been inverted and so have the relative transformations. Inversion of joint state variables is, of course, joint model-specific. We provide here an example of inverting the state (and acceleration) for three joint types:

- revolute joint → change sign of the angle,
- prismatic joint → change sign of the distance,
- spherical joint → invert the quaternion.

It is not as simple for the ${}_n i_{top}$ joint. Using the absolute-to-local approach again, however, we can find its joint-transform and then use it to perform an operation *inverse* to what *jcalc* does, i.e., derive this joint's local position state variables from the new spatial transform across it, ${}_n\boldsymbol{X}_{J_i}$.

First, we need to find ${}_n\boldsymbol{X}_{J_k}$, for $k = {}_n i_{top}$. By looking at Eqn. (8) simplified by the fact that we are considering the top joint (i.e., there is no parent transform to be concerned with), we can write

$$\boldsymbol{X}^{abs}_k = {}_n\boldsymbol{X}_{J_k}\,{}_n\boldsymbol{X}_{T_k}, \tag{21}$$

from which it follows that

$${}_n\boldsymbol{X}_{J_k} = \boldsymbol{X}^{abs}_k\,{}_n\boldsymbol{X}_{T_k}{}^{-1}. \tag{22}$$

Getting the raw joint position information from the spatial joint transformation matrix is (again) joint-type-specific. Below are some examples of what information must be extracted from ${}_n\boldsymbol{X}_{J_i}$ for certain joint models:

- revolute joint → angle of rotation about the joint local rotation axis,
- prismatic joint → amount of translation along the joint local translation axis,
- spherical joint → angles of rotation about the joint local axes.

7. Joint velocity and acceleration

The system whose hierarchy is being switched may have certain velocities and accelerations across its joints. What we do about them is very much task-specific and we will not provide a single, working approach since it does not seem to exist. In this section we shall present two sample scenarios and a way we handled velocity and acceleration when dealing with them.

7.1. Adapting motion capture data for ID. It is common to use different motion acquisition techniques to capture motion data and then provide it as an input to Inverse Dynamics (ID) algorithms in order to estimate the moments and forces governing the captured motion.

Standard humanoid motion capture data use a skeleton rooted somewhere at the hips/pelvis. While this is convenient and capable of creating a plausible motion, it is not really physically correct if we perceive the skeleton hierarchy as an order of physical support, so it cannot be used directly by ID.

What we can do is to apply the hierarchy switching algorithm frame-by-frame to alter the captured skeleton and motion sequence to make the actual support be the hierarchy root. A multi-support problem arises here, but it is beyond the scope of this article—let us assume we are dealing with single-support cases only.

Apart from the altered connectivity, bind pose and configuration, ID will obviously need joints' velocity and acceleration in each frame. We can, of course, numerically differentiate the position data after the hierarchy has been switched, but assuming that we already have high-quality derivatives (e.g., provided by the motion capture machinery), we can adapt them to the new hierarchy.

As usual, the procedure is different for non-top joints for which it is enough to *invert* the velocity and acceleration, which is a joint-type-specific task. Here are our standard examples:

- revolute joint → change the sign of angular velocity and acceleration,
- prismatic joint → change the sign of linear velocity and acceleration,
- spherical joint → change the sense of angular velocity and acceleration vectors,

and for the hierarchy top joint:

(i) total spatial velocity is found by accumulating the velocities of its old-hierarchy ancestors;

(ii) this total spatial velocity is then used to calculate the spatial velocity of the location chosen for the new top joint;

(iii) finally, the joint-specific velocity information is extracted from the obtained spatial velocity.

The procedure is exactly the same for the acceleration. Notice that A3 is not violated here since the procedure applied simply reevaluates the velocities and accelerations for the new hierarchy.

7.2. Hierarchy switch as a result of a collision. The second scenario is actually our use case. When a multibody system comes into contact with the environment, its support order changes, which must be reflected in the hierarchy. Multi-support can be an issue here, too, and again we have to assume that it is not (which is not an absurd assumption if you refer to the use case section). The proposed approach for this scenario is to detect collisions between the system and the environment (e.g., ground) while conducting a forward dynamics simulation. When a collision is detected, the hierarchy switch procedure is run with the collision point designated as the new support location.

The impact from the environment influences the velocities, and thus we need to derive and apply impact equations in order to step-change the velocities. There are several ways this can be approached, e.g., Mirtich (1996) integrated impact dynamics with Featherstone's AB algorithm. Another method, based on the conservation of angular momentum, is presented in the use case section of this article.

There is no need to track accelerations through the hierarchy switch/collision in this scenario.

It is hard to say whether A3 is kept here since the impact will most probably step-change the system velocities and thus modify its physical state. On the other hand, resolving the impact in a physically justified way cannot be perceived to violate the assumption of physical equivalence.

8. Sample use case

8.1. Passive dynamic walking. Passive dynamic walking is aimed at creating walking machines functioning without any actuation. McGeer, who is one of the

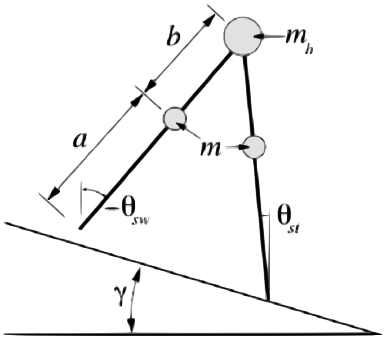

Fig. 7. Compass biped.

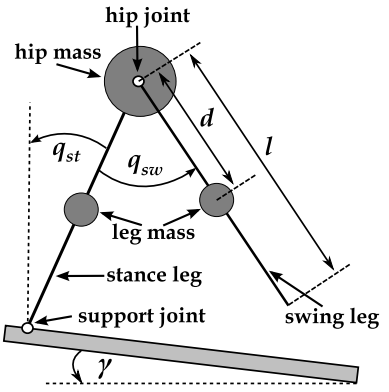

Fig. 8. Compass biped expressed in Featherstone's model terms. Both joints are of the revolute/hinge type.

pioneers in this field, proved that it is possible to construct a simplified, planar model of human legs which is able to walk stably down a gentle slope driven only by the constant gravitational field (McGeer, 1990).

A wide range of research groups are studying this kind of machines/robots analyzing systems of various levels of complexity—a straight-legged point-feet walker in 2D (Coleman, 1998; Garcia, 1999) and 3D (Coleman, 1998), a straight-legged curved-feet walker in 2D and 3D, curved-feet kneed walker in 2D (McGeer, 1990; Garcia, 1999), and more. One of the simplest cases among these is a planar compass-like biped (Fig. 7) which

- has no knees,
- has point feet.

We have decided to describe how one can simulate such a compass-like, planar walker using any of Featherstone's forward dynamics algorithms coupled with our base-switching algorithm and certain methodology to deal with feet-ground collisions.

8.2. Assumptions. The following assumptions about the model and its motion are made:

- **B1:** Leg-ground collisions are slipless plastic (no bounce, no slip).
- **B2:** Leg-ground collisions during the forward swing phase (*scuffing*) are ignored.
- **B3:** Leg-ground resting contact is slipless.
- **B4:** Stance and swing leg exchange roles instantaneously at the collision.
- **B5:** Joints are ideal (there is no friction or damping).

These assumptions are common for this kind of research.

8.3. Model. The *standard* model of a compass planar biped is depicted in Fig. 7. To express it using the terms described in the first part of this paper, we first identify its bodies and joints. From B5 we know that we can use ideal joints without introducing additional passive or active elements. B3 makes it possible to use a fixed base system with a revolute joint linking the ground with the support leg. B4 reassures us that no loops will be formed within the system (i.e., no double-support phase). Therefore, there are two bodies (two knee-less legs), and two joints (hip joint and support joint). The state of this system is fully given by the stance and swing leg angles, q_{st} and q_{sw}, respectively (defined as in Fig. 8), and their rates. The convention for defining stance and swing angles depicted in Fig. 7 is much more common and we are using a different set of symbols for the purpose of distinction (θ_{st} and θ_{sw} as opposed to q_{st} and q_{sw}).

Now, we need to find spatial inertia tensors for the bodies. Since masses in the system are lumped in three points while there are two bodies only, we have decided to make one of the legs consist of a single point mass placed somewhere along it, while the other is created by two point masses—one at its hip-end and the other at some distance along the leg. The resultant spatial inertia for the latter leg is constructed by adding matrices of inertia for the two point masses after having expressed the respective tensors in the common coordinate frame (latter body's frame).

The length of the legs is represented by mass-less rods joined at the hip and bearing the aforementioned masses lumped at certain points. These rods can collide with the ground but have no other physical meaning. The resultant model is depicted in Fig. 8.

The exact values of parameters used in the simulation are listed in Table 1.

8.4. Motion. The motion of this kind of systems is *hybrid* in the sense that it is characterized by continuous dy-

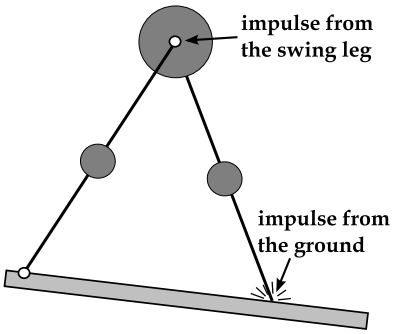

Fig. 9. System at the moment of the swing leg colliding with the ground.

namics interleaved by discrete events (ground collisions) (Hiskens, 2001). Thus, in the interval between two consecutive ground collisions we apply forward dynamics algorithm to the model described in the previous subsection subject to a constant gravitational field, which is equivalent to a double pendulum simulation.

When a collision occurs, a special treatment is needed that takes into account the impulsive nature of this event. This special treatment provides us with new, post-impact velocities that can be used as an initial condition for the upcoming continuous dynamics phase.

8.5. Collision resolution. Collisions are resolved based on B1 and the principle of conservation of the angular momentum. The system configuration is preserved throughout the collision. Moreover, by looking at the system during a collision (Fig. 9), we can see that no impulsive torque is applied to the entire system about the new contact point and to the sub-system comprising the pre-impact swing leg about the hip. Therefore, angular momenta about the contact point and the hip will be conserved through the collision for the entire system and the former swing leg, respectively. The conservation of the angular momentum leads to a step change of system angular velocities.

Based upon these two observations, we can provide two equations in two unknowns (post-impact velocities) by simply equating the preserved angular momenta expressed in pre- and post-impact terms.

To keep things as general as possible, we have decided to use an approach proposed by Garcia (1999), who used the generalized inertia matrix of the system to build the aforementioned set of equations and then solved it numerically. Finding symbolic versions of these equations is not particularly difficult in this case but would have to be done separately for different models and probably offline. Last but not least, if the forward dynamics algorithm applied happens to be CRBA-based (or equivalent), then the generalized inertia matrix for the system is already available *for free*.

Table 1. Simulation parameters and initial conditions (see Fig. 8 for reference).

Parameter	Value	Unit
m_h (hip mass)	2	kg
m_l (leg mass)	1	kg
l	1	m
d	0.5	m
γ	0.0524	rad
q_{st}	-0.3236	rad
q_{sw}	0.5424	rad
$\dot{q}_{st}$	-1.4939	rad/s
$\dot{q}_{sw}$	-0.3117	rad/s

The two equations in the matrix form are as follows:

$$\begin{bmatrix} {}^{-}L_{tot} \\ {}^{-}L_{sw} \end{bmatrix} = \boldsymbol{H} \begin{bmatrix} {}^{+}\dot{q}_{st} \\ {}^{+}\dot{q}_{sw} \end{bmatrix}, \tag{23}$$

where the minus in the superscript stands for pre-impact, the plus for post-impact, $\boldsymbol{H}$ is a 2×2 generalized inertia matrix of the system, L_{tot} is the total angular momentum of the system about the contact point, L_{sw} is the angular momentum of the pre-impact swing leg about the hip, $\dot{q}_{st}$ and $\dot{q}_{sw}$ are the angular velocities of the post-impact stance and swing legs, respectively. Angular momenta and angular velocities are scalars since this is a planar case.

8.6. Initial conditions and system parameters. In order to achieve a stable gait, beside proper simulation and proper impact calculations, one also needs to specify initial conditions for specific model parameters and slope. These conditions are the model pose (joints angles) and angular velocities (hinge joints velocities) that we provide the simulation with at launch.

We have decided to refer our simulation directly to someone more adept in this type of research and used initial velocities and angles provided by Hiskens (2001) along with exact system parameters values. They are collectively presented in Table 1 together with system parameters.

8.7. Results. Using the above-described methodology, parameters, initial conditions and the 4-th order Runge–Kutta integrator, we have managed to obtain a steady gait. We have assessed it in most natural terms by observing the rendered motion. To provide an additional illustration, we employ a phase portrait for the system, which is, however, four-dimensional, making it hard to visualize directly. This problem is commonly solved by projecting the entire phase portrait onto a plane representing one leg (Fig. 10). Which leg we choose is unimportant since the steady gait is symmetric.

Obviously, such a projection loses vital pieces of information—non-cycles can appear as cycles and differ-

ent trajectories may cross; nonetheless, this is a standard procedure in this kind of research.

We will now briefly analyze the phase portrait depicted in Fig. 10. Two continuous parts can be distinguished, the support and the swing phase, separated by discontinuities in the velocity. The continuous parts are the effect of applying a general rigid body system simulation procedure. When a collision condition is met, an impact resolution technique is applied, which results in a step change of the velocities and in switching of roles that each leg plays in the system—the former support leg becomes a stance leg and vice versa. This is why the contact events (contact gain and loss) initiate the continuous phases in the plot. There are no discontinuities of the position variable/angle since the system configuration is preserved through the collision.

As pointed out by Goswami (1996), a significant amount of insight into the nature of the symmetric passive gait may be gained by plotting the potential energy of a walking biped as a function of its kinetic energy. Figure 11 provides such a plot using the results of our simulation. While steadily walking the biped will follow a *path* given by $ABAC$—it will first gain potential energy (AB), which will then be transformed into kinetic energy (AC, building up velocities). In Fig. 11 these energy transformations happen along a fixed line since the total energy of the system is constant. However, when a ground collision occurs (C), a specific amount of system energy is lost (CA'), which drives the system to a parallel line representing a lower total energy. This scenario is repeated in every step of a steady gait.

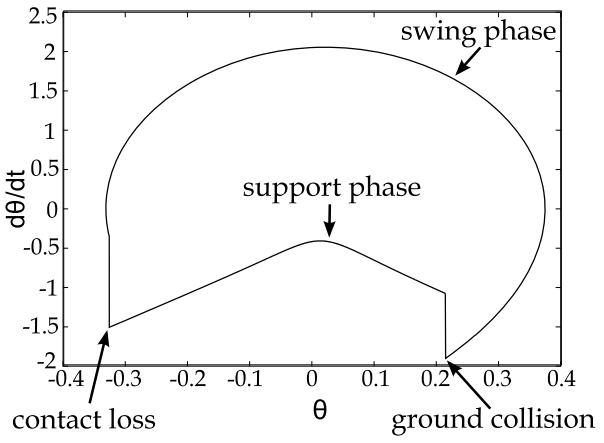

Fig. 10. Phase portrait projected onto the $\theta\dot{\theta}$ plane (θ is the angle made by the analyzed leg with the vertical). We can distinguish two continuous parts—the top one corresponds to the period when the leg in question is swinging, while the bottom one represents the period when it is supporting.

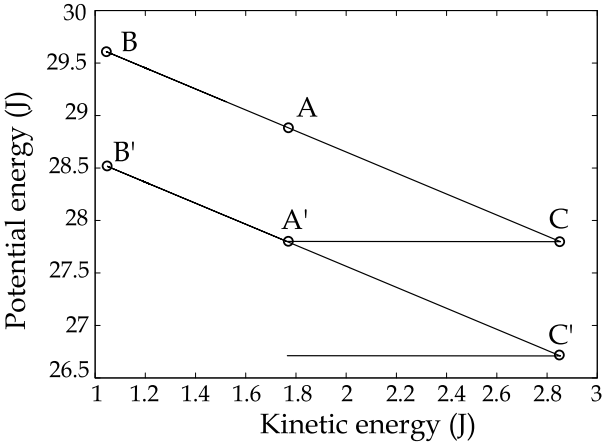

Fig. 11. Biped potential energy plotted as a function of its kinetic energy for two consecutive steps. Points of interest have been marked and labeled as follows: A—starting point, B—maximum potential energy, C—ground collision. Prime symbols indicate the second step.

9. Conclusions

In this paper we provided an algorithm capable of rebuilding the entire rigid body system definition, which will work with arbitrarily configured kinematic trees expecting minimal information from the user/caller. Apart from the system configuration, we also proposed ways of modifying velocities and, if necessary, accelerations of bodies forming the system, depending on the application.

The proposed algorithm, used along with an appropriate impact resolution technique, was successfully applied in the sample usage scenario the results of which were verified in several ways. Our research also proved that algorithm-based simulation along with auxiliary algorithms, such as the one described in this paper, can lead to development of powerful simulation tools.

Acknowledgment

This paper has been partly supported by the project *System with a library of modules for advanced analysis and an interactive synthesis of human motion* co-financed by the European Regional Development Fund under the Innovative Economy Operational Programme (Priority Axis 1, Measure 1.3, Submeasure 1.3.1).

References

Baraff, D. (1996). Linear-time dynamics using Lagrange multipliers, *SIGGRAPH '96: Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, New York, NY, USA*, pp. 137–146.

Chace, M. and Sheth, P. (1973). Adaptation of computer techniques to the design of mechanical dynamic machin-

ery, *Design Engineering Technical Conference, Cincinnati, OH, USA*, ASME Paper 73-DEPT-58.

Coleman, M. (1998). *A Stability Study of a Three-Dimensional Passive-Dynamic Model of Human Gait*, Ph.D. thesis, Cornell University, Ithaca, NY.

Craig, J. (2005). *Introduction to Robotics: Mechanics and Control, 3rd Edition*, Prentice Hall, Upper Saddle River, NJ.

Featherstone, R. (1983). The calculation of robot dynamics using articulated body inertias, *International Journal of Robotics Research* **2**(1): 13–30.

Featherstone, R. (1984). *Robot Dynamics Algorithms*, Ph.D. thesis, Edinburgh University, Edinburgh.

Featherstone, R. (1987). *Robot Dynamics Algorithms*, Kluwer Academic Publishers, Boston, MA/Dordrecht/Lancaster.

Featherstone, R. (2008). *Rigid Body Dynamics Algorithms*, Springer, New York, NY.

Garcia, M. (1999). *Stability, Scaling, and Chaos in Passive-Dynamic Gait Models*, Ph.D. thesis, Cornell University, Ithaca, NY.

Goswami, A., Thuilot, B. and Espiau, B. (1996). Compass-like biped robot, Part I: Stability and bifurcation of passive gaits, *Research Report RR-2996*, INRIA, Montbonnot Saint Martin.

Hiskens, I. (2001). Stability of hybrid system limit cycles: Application to the compass gait biped robot, *Proceedings of the 40th IEEE Conference on Decision and Control, Orlando, FL, USA*, pp. 774–779.

Hooker, W. and Margulies, G. (1965). The dynamical attitude equations for an n-body satellite, *Journal of the Astronautical Sciences* **12**(4): 123–128.

Jain, A. (2011). *Robot and Multibody Dynamics: Analysis and Algorithms*, Springer, New York, NY/Dordrecht/Heidelberg/London.

McGeer, T. (1990). Passive dynamic walking, *International Journal of Robotics Research* **9**(2): 62–82.

Mirtich, B. (1996). *Impulse-based Dynamic Simulation of Rigid Body Systems*, Ph.D. thesis, University of California at Berkeley, CA.

Orlandea, N., Chace, M. and Calahan, D. (1977). A sparsity-oriented approach to the dynamic analysis and design of mechanical systems, Part 1, *Transactions of the ASME Journal of Engineering for Industry* **99**(3): 773–779.

Paul, B. (1975). Analytical dynamics of mechanisms—A computer oriented overview, *Mechanisms and Machine Theory* **10**(6): 481–507.

Roberson, R. and Wittenburg, J. (1966). A dynamical formalism for an arbitrary number of interconnected rigid bodies with reference to the problem of satellite attitude control, *Proceedings of the 3rd International Federation of Automatic Control Congress, London, UK,* pp. 46D.2–46D.9.

Rodriguez, G. (1991). A spatial operator algebra for manipulator modeling and control, *International Journal of Robotics Research* **10**(4): 371–381.

Uicker, J. (1965). *On the Dynamic Analysis of Spatial Linkages Using 4 by 4 Matrices*, Ph.D. thesis, Northwestern University, Evanston, IL.

Vereshchagin, A. (1974). Computer simulation of the dynamics of complicated mechanisms of robot manipulators, *Engineering Cybernetics* **12**(6): 65–70.

Walker, M. and Orin, D. (1982). Efficient dynamic computer simulation of robotic mechanisms, *Transactions of the ASME Journal of Dynamic Systems, Measurement and Control* **104**(3): 205–211.

Wittenburg, J. (2007). *Dynamics of Multibody Systems*, Springer, Berlin/Heidelberg/New York.

Jakub Stępień received his M.Sc. degree in computer science from the Silesian University of Technology in 2008. He is currently a Ph.D. student at the Institute of Informatics of the same university. His research interests include human motion analysis and synthesis, physical simulation and computer animation.

Andrzej Polański received his M.Sc. (in electronic engineering), Ph.D. and D.Sc. (in automatic control) degrees from the Silesian University of Technology in 1982, 1990 and 2000, respectively. He obtained his titular professorship (state degree) in technical sciences in 2009. At present he is a professor at the Institute of Informatics, Silesian University of Technology, and at the Polish-Japanese Institute of Information Technology. His scientific interests involve systems theory, problems of stability and control of dynamic systems, as well as various areas of mathematics applied in biology and medicine.

Konrad Wojciechowski received the M.Sc. diploma in electrical engineering from the Academy of Mining and Metallurgy, Cracow, Poland in 1967, and the Ph.D. and D.Sc. degrees in control theory from the Silesian University of Technology, Gliwice, Poland, in 1976 and 1991, respectively. He received the professorial title in 1999. His area of scientific activity covers linear and nonlinear control theory, neural nets, image processing and pattern recognition, computer vision, computer graphics, animation and games. He has published 180 papers in refereed journals and conference proceedings on control theory, image processing and computer vision. He has been a supervisor of 30 Ph.D. dissertations in the field of computer vision and graphics, and a reviewer of over 20 such theses.

Received: 28 March 2011
Revised: 11 July 2011